Ontology-Based Discovery of Data-Driven Services

Maarten Bynens, Bart De Win and Wouter Joosen
Department of Computer Science - DistriNet
Katholieke Universiteit Leuven
Celestijnenlaan 200A
Leuven, Belgium
Email: maartenb, bartd, wouter@cs.kuleuven.be

Bart Theeten
Alcatel Bell
Francis Wellesplein 1
Antwerp, Belgium
Email: bart.theeten@alcatel.be

Abstract—Current service technologies are primarily focused on the functionality of services. A significant portion of the available services, however, exhibits a data-driven rather than a functionality-driven character, which makes the current technologies less appropriate. This paper focuses on discovery for data-driven services as part of data federation as an overall goal. The primary requirements and characteristics are discussed and a prototype implementation based on ebXML is presented. Although significant progress has been made, many practical issues remain to be addressed in order to get this model fully operational.

I. INTRODUCTION

Services in the context of service-oriented architectures, or more specifically web services, are typically characterized by the functions they support. The development and use of services is *functionality-driven*: services are defined, searched for and connected with based on their functionality, using WSDL, UDDI and SOAP(-RPC) respectively. For some types of services however, the functionality closely resembles the management of the service's data. Most operations of a typical calendar service, for instance, are concerned with data management rather than with functionality (apart from perhaps group scheduling, which is quite complex). These services are *data-driven* rather than functionality driven. Recently, the data-driven approach for services is gaining importance, illustrated, for instance, by many on-line services providing a Representational State Transfer (REST) API [5], which favors this approach.

Services in the context of our work are considered to be deployed on an enterprise service bus (ESB). An ESB is a distributed and standards-based integration platform that foresees in messaging, intelligent routing and transformation capabilities to reliably connect and coordinate the interaction of services. In such a setting there is a need to focus on both the available data and the available functionality. The management of data available on a service bus introduces different kinds of problems: (i) data is spread out, and often duplicated, between the services registered on the bus; services manipulate similar data that resides at different locations and, hence, synchronization of these (semantically equivalent) data items is an issue; and (ii) the data models of interacting services are not compatible and need to be bridged.

Service-level data federation can address these issues. The concept of data federation has already been studied extensively

for database management systems [6]. In the context of service-oriented architectures, the main advantages of a data federation include:

- Mediation between services: the services on the bus are provided by a third party and are deployed without a priori agreements. As a result, the services do not conform to a common data model (see also [7]). Because of this, the data federation will operate as a mediator between these services.
- Data-driven composition: besides the explicit functionality-driven composition of services, the discovery and composition of services can be driven by the relationships between their data models. Client services could look for data in which they are particularly interested.

In the context of ESB, a dedicated Federated Data Manager (FDM) can significantly help to realize this data federation model. Conceptually, a FDM can be thought of as providing different services, including: *Discovery* to locate the data available on the bus and maintaining a model that represents this data, *Query* to support integrated queries that search over different services and data models, *Provisioning* to provide the data for newly registered services based on data already available on the bus and *Synchronization* to keep similar data in a consistent state. In this paper we focus on the characterization and realization of data discovery in particular.

Traditional service discovery (e.g., as provided by UDDI[8]) enables businesses to publish service listings and discover services from other businesses. The meta data available in the registry to describe and search for services is rather limited and mainly concerns businesses, protocols and standard classifications. In the light of data-driven services, this discovery functionality is not sufficient. The main contributions of this paper are the discussion of the characteristics of data-driven service discovery and the presentation of a general discovery service model (and concrete prototype implementation).

The remainder of this paper is structured as follows. In Section 2, the requirements for data-driven service discovery are sketched. Section 3 discusses the functionality of a discovery service, whose realization is further presented in Section 4. Section 5 discusses a number of open issues and, finally, Section 6 concludes.

II. REQUIREMENTS FOR DATA-DRIVEN SERVICE DISCOVERY

The discovery service is the backbone of a FDM. It is responsible for discovering and locating services and their data usage. The key characteristics and requirements for a data-driven discovery service are:

- The basis of the discovery service is a registry that stores the data models of all the registered services.
- The discovery service must be able to inspect the interface of a service (or any other specification for that matter) and infer the data model from this description.
- For service mediation, the FDM should understand the semantic differences between related data models. For instance, the order of attributes in two related types may differ, or they may be represented as separate versus aggregate attributes. This can be represented by means of an ontology encompassing *relationships* (to represent relations within a single data model) as well as *associations* (to represent semantic relations between different data models). In this work, we will use a global ontology that represents all data types in a single model.
- When a new service is registered, the discovery service should update the registry, and discover and instantiate new relationships and associations with the service's data model (also known as matching [9]).
- In addition to the associations between different data types, the discovery service should incorporate knowledge of how to transform between these data types. This could be achieved by associating every data relationship with (a reference to) a transformation service, which is able to transform one data type in the relationship to the other and/or vice versa.
- The resulting ontology should be able to transform between any two related data types, which may require a chain of associations and, hence, a chain of transformations. Therefore, the discovery service should be able to deduce how to map one service to another using these transformations. In this paper, we will use the term *route* for a chain of transformations. The primary use of those routes is the autonomous synchronization between services.
- Metadata can be linked with data types to provide extra information on these types. This enables semantic data discovery, like locating a service that deals with multimedia content rather than just looking for movies or books.

III. FUNCTIONALITY OF DATA-BASED DISCOVERY

The functionality of data-based discovery consists of three major activities: extracting the data model from the interface of registering services; relating the extracted data model to the ontology stored in the registry and querying the stored ontology to discover services based on their data model. The following paragraphs elaborate on these activities.

Data model extraction: When a new service is registered at the discovery service, the interface of the service will be inspected to extract a new data model. In the current state

of the practice, services are unaware of data federation. The significance of the data part on the interface will be small and the information the discovery service will be able to extract, is rather limited. For instance, a WSDL description usually contains only a basic description of the data types used on the input or output of the operations of a service.

Data model integration: The integration of the service's data model in the currently stored data model boils down to (i) distinguishing between new and already existing data types and (ii) identifying relationships between new data types and previously known data types. While the former is quite straightforward, the latter is more complex.

We strive for automatic integration of new data models in the global ontology, although the state of the practice makes this task very difficult. The discovery service will suggest new associations between data models based on the names of the types and relations in those models. Afterwards, a discovery operator will have to finalize these suggestions manually and provide the necessary transformations to accompany these associations.

Querying the registry's ontology: We foresee that the primary use of the discovery service is to search for related services. For the discovery service to be able to search for related services through their data models, it needs some rules to define which relations at the level of the data model can introduce relations at the level of services.

Informally, we can state that two services are related if the inputs of an operation of one service can be transformed to provide the inputs of an operation of the other service. The inputs can be transformed if there exists a route of associations and relations with corresponding transformations. A route of transformations can consist of three kinds of transformations, respectively corresponding with an association, the inverse of a *partOf* relation and a *partOf* relation in case it is the only part.

IV. PROTOTYPE

This section elaborates on our prototype implementation of the functionality sketched above. The foundations of our prototype can be summarized as follows:

- data extraction based on WSDL[11], optionally with a separate (semantic) data interface (OWL);
- a registry based on ebXML, with the necessary extensions for data association and semantic discovery.

Regarding the data discovery technology, we choose ebXML[4] over UDDI, since it offers a much more expressive data model and query API. Standard technologies are being used as much as possible. WSDL and OWL can be considered well known, but maybe this is less the case for ebXML. Therefore, we give a brief introduction in the next section.

A. ebXML in a nutshell

ebXML is a set of specifications for electronic business collaboration, of which discovery is one part. Most important to us is the registry used by ebXML. An ebXML registry consists of both a registry and a repository. The repository is

Fig. 1. High level architecture of discovery service

capable of storing any type of electronic content, while the registry is capable of storing meta data that describes that content. The ebXML registry defines a Registry Information Model (RIM) which specifies the standard meta data that may be submitted to the registry. The main features of the information model include:

- **RegistryObject** The top level class in ebRIM, it is an abstract base class used by most classes in the model. It provides minimal meta data for registry objects.
- **Classification** Any RegistryObject may be classified using ClassificationSchemes and ClassificationNodes. A ClassificationScheme defines a tree structure made up of ClassificationNodes
- **Association** Any RegistryObject may be associated with any other RegistryObject using an Association instance where one object is the sourceObject and the other is the targetObject of the Association instance. An Association instance may have an associationType which defines the nature of the association.
- **Service Description** The Service, ServiceBinding and SpecificationLink classes provide the ability to define service descriptions including WSDL.

ebXML exports two interfaces to use the registry. The LifecycleManager (LCM) is responsible for all object lifecycle management requests. The QueryManager (QM) is responsible for handling all query requests. A client uses the operations defined by this service to query the registry and discover objects. All information can be used to search for items in the registry, e.g. all RegistryObjects that are associated with a certain item or all Service items that are classified with a certain ClassificationNode.

To enhance the data classification model in the ebXML registry with semantic relationships, the constructs available in ebXML can be used. The ebXML registry information model can be used to simulate an OWL description of data classes, as described in [1].

B. Architecture of the data discovery service

An architecture has been defined for the data discovery service prototype using ebXML as a backbone component. Figure 1 depicts a high level component view of the architecture. It consists of three components:

1) *Discovery component*: This component provides three interfaces: Lifecycle, Admin and Query. Lifecycle is used for the management of the registered services. It is used by the discovery operator to subscribe, publish and activate new services. Maintenance operations on the registry, especially on the data models and the associations between them, are

available on the Admin interface. The Query interface, lastly, is used for searching the information stored in the registry. It offers one specific operation to find routes to related services, and one generic operation for SQL-like queries as defined in the ebXML standard.

2) *ebXML component*: A fully ebXML standard compliant registry and discovery service. It will be used by both the discovery service and third-party clients. The former will use it as a registry that stores the available services together with their data models, the global ontology and associations between them, while the latter can use it as a traditional discovery service.

3) *QueryFacade component*: This component will handle recursive queries, for example to search through transitive relations. This component is necessary because the ebXML standard specification does not include this functionality. For instance, searching for all the parts of a certain item, is accomplished by querying for the parts of the item and repeatedly querying the registry for the parts of those parts.

C. Implementation of the functionality

The interfaces of the discovery service mainly use WSDL and OWL formats as input and output, but internally, the discovery registry is based on the ebXML format. Extraction of the data model will thus come down to transforming WSDL and OWL to the ebRIM[2] and ebRS[3] publication format. Some mappings we use are shown in figure 2. An example

Data Model	ebRIM
Service	Service
Service information	ServiceBinding
WSDL description	SpecificationLink
Service data model	ClassificationScheme
Data type	ClassificationNode
Relating service and data	Classification

Fig. 2. Mappings used by the prototype

of how the prototype could be used, is the synchronization between services with overlapping data models. Consider, for example, an addressbook and an instant messaging service. A client could wish to change the address of one of the entries in the addressbook. The instant messaging service, in its turn, stores a collection of Vcards, which also contain address information. In a data federation environment, it could become possible that when the address of an addressbook entry is about to change, a corresponding Vcard in the instant messaging service is updated as well.

To accomplish this, the data models of these services are associated and stored in the registry. The discovery service will automatically deduce a set of suggested datatype relationships, to be finalized by the discovery operator. This automatic deduction is heuristics based, but the details of the approach are out of scope for this paper. For example, figure 3 shows how the addressbook service could be stored in the ebXML registry. The service is classified with two datatypes, one for changing address information and another for adding new entries on the addressbook (the operations themselves have been omitted from the figure). As you can see these types consist of an address type, a person type and strings. To

Fig. 3. Meta data of the AddressBook service as stored in the ebXML Registry

keep the figure manageable, some simplifications are made with regard to the associations. A white diamond denotes an Association of type *ObjectProperty* and a black diamond denotes an Association of type *DataProperty* (as in OWL). The diamond-side corresponds to the *SourceObject*, while the other side corresponds to the *TargetObject*.

For the seamless synchronization of these services, at run-time, the discovery will search for a route of transformations between them. A possible route in the example could look like this: from the *UpdateAddress* operation to the *UpdateVCard* operation via the transformations that map *UpdateAddress* to the *Address* data type, the *Address* data type to the address type as it is used in the *VCard* data type and from there, via *VCard* to *updateVCard*.

V. DISCUSSION

Quite some decisions in our approach stem from pragmatical reasons, but when the state of the practice evolves, the functionality of data-based discovery can be refined and extended. Adding semantic information to both functionality and data types in the interface of a service, is an evolution which has already started. Both the integration of new data types as searching for routes of transformations can benefit from this. Data types on an interface can be described semantically (e.g. using inlined OWL constructs [10], or using a separate OWL file) by relating the types to other, known, types or integrating them in a common or standard ontology. This will result in a more meaningful global ontology in the registry.

Our prototype implementation has been tested on limited example services, among others the examples illustrated in the previous section. Although we are convinced of the applicability of our approach for larger systems, we do expect that for our current model a number of issues will arise in practice. Firstly, our current data model is based on associations and transformations between types. In practice however, we expect that the interfaces of services will not be sufficiently strongly typed to prevent that equivalent types are used differently. For example, the basic data type *string* can be used for a lot

of things, like, a street name, an email address, a password, and so forth. Non-basic types also exhibit this problem. A possible solution to this issue would be to use the semantics of the relations between types instead of the semantics of the types themselves. For instance, we could include knowledge of the fact that the first *String* field in the *Address* data type is the street information. We are now extending our system to support this functionality.

Secondly, we cannot prevent the presence of dynamic data types. For example, the data type *address* could have a field that specifies whether it is a personal address or a business address. Dealing with such types needs run-time support, because the exact semantics cannot be determined up front.

VI. CONCLUSION

This paper has presented a discovery service for data-driven services. The discovery service is part of a larger data federation approach for web services that enables data-driven composition and synchronization of services. The fundamental activities of the discovery service are (1) the extraction of the data model from a WSDL description, (2) associating data models of newly subscribed services with the global ontology stored in the registry and (3) searching through the stored information to find services with overlapping data models. We have prototyped the discovery service using ebXML as the underlying technology.

This work is not finished and the authors see many ideas for improvements and extensions, including the use of OWL as a semantic data description (possibly in a separate data specification), the extension of the integration activity to stretch the spectrum of automatic integration activities by using better heuristics on data type relationships, the extension of the query activity to take into account relationships as well, finding solutions for the open issues regarding the data model, and so forth. In general, the current focus on functional services makes this data-driven approach harder than it should be, so improvements in that area will definitely benefit this work as well.

REFERENCES

- [1] Asuman Dogac, Yildiray Kabak, Gokce Laleci, Carl Mattocks, Farrukh Najmi, and Jeff Pollock. Enhancing ebxml registries to make them owl aware. *Distributed and Parallel Databases*, 18(1):9–36, 2005.
- [2] ebXML. ebxml registry information model specification.
- [3] ebXML. ebxml registry services.
- [4] ebXML. Electronic business with xml, <http://ebxml.org/>.
- [5] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.
- [6] David K. Hsiao. Federated databases and systems: Part i - a tutorial on their data sharing. *VLDB J.*, 1(1):127–179, 1992.
- [7] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, John A. Miller, and Jonathan Lathem. Semantic interoperability of web services - challenges and experiences. In *2006 IEEE International Conference on Web Services (ICWS 2006)*, 2006.
- [8] OASIS. Universal discovery and description integration, <http://www.uddi.org/>.
- [9] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
- [10] W3C. Web ontology language, <http://www.w3.org/2004/owl/>.
- [11] W3C. Web service description language, <http://www.w3.org/tr/wsdl>.